四月七号 前三天的笔记

Pythonanywhere部署

<http://yaoyongkang.pythonanywhere.com/>

git相关

git --version 查看git是否安装

git全局设置

git config --global user.name 'yyk'

git config --globla user.email '2670211238@qq.com'

git初始化本地仓库

git init

新建.gitignore文件

nano.gitignore

\*.pyc

\*~

\_\_pycache\_\_

.vscode

删除上传的文件

git rm -r --cached day1\watchlist\env

git commit -m "删除env"

git push

本地仓库和远程仓库连接

检查是否已经创建了SSH密钥 cat ~/.ssh/id\_rsa.pub

如果上一步显示“No such file or directory”，这说明之前你没有生成过这个ssh密钥，需要先生成，使用ssh-keygen，不管提示任何东西，一路enter

指定远程仓库地址（绑定远程仓库） git remote add origin git@github.com:ITClubOfBruce/flaskweb.git

安装虚拟环境

python -m venv '虚拟环境名字'

激活虚拟环境 windows上 : env\Scripts\activate

Mac上: . env/bin/activate

widnows上vscode的终端中激活：. env\\\Scrpits\\\activate（注意加点空格）

第一次提交

git status 查看flaskweb文件夹下有哪些文件发生了变化 （第一次提交时用，以后可以不用）

git add . 追踪变化的文件

git commit -m "第一次提交，项目初始化完成" 提交到本地仓库

git push -u origin master "-u"参数会将推送到目标仓库和默认分支

运行本地开发服务器

flask run

Environment: production //默认启动的时候是生产环境 （必须改成开发环境）

Debug mode: off //调试模式，开发阶段一定要是on的状态

修改生产环境

#### 使用python-dotenv管理系统环境变量

安装 pip install python-dotenv

python-dotenv会从项目根目录的 .flaskenv和 .env文件读取环境变量并设置。我们需要创建这两个文件 touch .flaskenv .env

.flaskenv ：用来存储Flask命令行系统相关的公共环境变量 FLASK\_ENV = development

.env：存储敏感数据，这个文件不能被提交到github

## 五、URL

1. 多URL：一个视图函数可以绑定多个URL

@app.route('/')

@app.route('/index')

@app.route('/home')

def index():

return "<h1>Hello Flask llaall </h1>"

2. 动态URL

@app.route('/user/<name>')

def index(name):

return "<h1>Hello %s </h1>"%name

3. url\_for()

修改路由

生成URL：第一个参数是端点值，默认是视图函数的名称，静态资源默认是static

特点：在模板中可以直接使用`<img src={{ url\_for('static',filename='static/images/avatar.jpg') }}>`但是在python文件中，需要先导入url\_for

## 六、模板temlates

模板：包含了变量和运算逻辑的HTML或者其他格式的文本

Jinja2完成渲染工作的

{{ }} 标记变量

{% %} 标记语句

{# #} 注释

## 七、 静态资源

<!-- 导入css样式 -->

<link rel="stylesheet" href="{{url\_for('static',filename='css/aa.css')}}">

导入图片

 <img src="{{url\_for('static',filename='img/5.jpg')}}" alt="" class="tp">

## 八、数据库操作

ORM：对象关系映射

模型类：借助SQLAlchemy，通过定义Python类来表示数据库里的一张表（类的属性表示表中的字段/列），通过这个类进行各种操作来代替写SQL语句，这个类就是模型类 FLask-SQLAlchemy

1. 安装 pip install flask-sqlalchemy

2. 导入扩展类，初始化实例

from flask\_sqlalchemy import SQLAlchemy

app = Flask(\_\_name\_\_)

db = SQLAlchemy(app)

3. 设置数据库URI

Flask提供了一个统一的接口来写入和获取这些配置变量：Flask.config字典，配置变量的名称必须使用大写，写入配置的语句一般会放到拓展类实例化语句之前

4. 创建模型类

# models 数据层

class User(db.Model):

id = db.Column(db.Integer,primary\_key=True)

name = db.Column(db.String(20))

class Movie(db.Model):

id = db.Column(db.Integer,primary\_key=True)

title = db.Column(db.String(60))

year = db.Column(db.String(4))

db.Integer 整型 db.Text 长文本 db.DateTime 时间日期

db.String(size) 字符串 db.Float() 浮点数 db.Boolean 布尔值

primary\_key=True 设置当前字段是否为主键

nullable 布尔值，是否允许为空值

index 布尔值，是否设置索引

unique 布尔值，是否允许重复值

default 默认值

5. 创建数据表

flask shell

from app import db

db.create\_all()

如果你改动了这个模型类，想重新生成表模型，需要先使用db.drop\_all()删除表，然后重新创建

flask initdb 初始化数据库

flask initdb --drop 删除之后再创建

# 自定义命令

@app.cli.command() # 注册为命令

@click.option('--drop',is\_flag=True,help="先删除再创建")

def initdb(drop):

if drop:

db.drop\_all()

db.create\_all()

click.echo("初始化数据库完成")

6. 数据库的增删改查

增加

>>> from app import db

>>> from app import User,Movie

>>> user = User(name='Bruce')

>>> movie1 = Movie(title='杀破狼',year='2000')

>>> movie2 = Movie(title='战狼',year='2016')

>>> db.session.add(user) //把新创建的记录添加到数据库会话

>>> db.session.add(movie1)

>>> db.session.add(movie2)

>>> db.session.commit() //提交数据库会话，只需要在最后执行一次即可

```

读取

模型类.query.过滤方法.查询方法

>>> Movie.query.filter\_by(title='战狼').first()

>>> Movie.query.filter(Movie.title=="战狼").first()

| 过滤方法 | 说明 |

| ----------- | -------------------------------------------- |

| filter() | 使用指定的规则过滤记录，返回新产生的查询对象 |

| filter\_by() | |

| order\_by() | 根据指定条件对记录进行排序 |

| group\_by() | 分组 |

| 查询方法 | |

| -------------- | ---------------------------- |

| all() | |

| first() | |

| get(id) | |

| count() | 返回查询结果的数量 |

| first\_or\_404 | |

| get\_or\_404(id) | |

| paginate | 返回一个Pagination对象，分页 |

更新

```

>>> from app import db,Movie

>>> movie = Movie.query.get(2)

>>> movie.title

'战狼'

>>> movie.title="战狼2"

>>> movie.year

'2016'

>>> movie.year = '2018'

>>> db.session.commit()

```

删除

```

>>> movie = Movie.query.get(1)

>>> db.session.delete(movie)

>>> db.session.commit()

```

## 九、自定义错误页面，错误处理函数，模板继承，模板上下文处理函数

1. 自定义错误页面

2. 错误处理函数

```

# 错误处理函数

@app.errorhandler(404)

def page\_not\_found(e):

user = User.query.first()

return render\_template('404.html',user=user)

```

3. 模板上下文处理函数

```

# 模板上下文处理函数

@app.context\_processor

def common\_user():

user = User.query.first()

return dict(user=user)

```

这个函数的返回值是一个字典，会统一注入到每一个模板的上下文环境中，因此可以在模板中直接使用

4. 模板继承

先建立一个base模板（基础模板）

head标签里面添加

<head>

    {% block head %}

    <meta charset="UTF-8">

    <meta name="viewport" content="width=device-width, initial-scale=1.0">

    <title>{{user.name}}</title>

<!-- 导入css样式 -->

<link rel="stylesheet" href="{{url\_for('static',filename='css/aa.css')}}">

    {% endblock %}

</head>

<body>

 {% block content %} {% endblock %}

</body>

在别的模板调用base,html

{% extends 'base.html' %}

{% block content %}

    <p>总共{{movies|length}}</p>

    <ul class='movie-list'>

      <li>页面走丢了~~~~404

          <a href="{{url\_for('index')}}">返回主页</a>

      </li>

    </ul>

{%endblock%}

## 十、表单

1. 报错

```

Method Not Allowed

The method is not allowed for the requested URL.

```

原因：我们的表单发送的是POST请求，但是视图函数默认只能处理GET请求

修改视图函数 @app.route('/',methods=['GET','POST'])

GET：返回渲染后的页面

POST：获取表单传递过来的数据

2. 验证数据，给用户以提示信息

flask中的flash模块来做提示，在python文件中通过flash('输入错误')，通过get\_flashed\_messages()来展示到模板

```html

<!-- flash消息 -->

{% for message in get\_flashed\_messages() %}

<div class="alert"> {{ message }} </div>

{% endfor %}

<nav>

```

3. 报错：The session is unavailable because no secret key was set. Set the secret\_key on the application to something unique and secret.

flash()函数在内部会把消息存储到Flask提供的session对象中。session用来在请求间存储数据，它会把数据签名后存储到浏览器的Cookie中，所以我们需要设置前面所需的密钥

app.config['SECRET\_KEY'] = '1903\_dev'